22

(51) International Patent Classification⁷: G06F 9/44

(21) International Application Number: PCT/US02/19883

(22) International Filing Date: 20 June 2002 (20.06.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/885,836     20 June 2001 (20.06.2001)     US

(71) Applicant (for all designated States except US): SAP AK-TIENGESELLSCHAFT [DE/DE]; Intellectual Property Department, Neurottstr. 16, 69190 Walldorf (DE).

(72) Inventor; and
(75) Inventor/Applicant (for US only): DHARAMSHI, Gautam [US/US]; 2330 California Street #5, Mountain View, CA 94040 (US).

(74) Agent: KIRKLAND, Mark, D.; Fish & Richardson P.C., 500 Arguello Street, Suite 500, Redwood City, CA 94063 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— with international search report

[Continued on next page]

(54) Title: JAVA RULE ENGINE FRAMEWORK
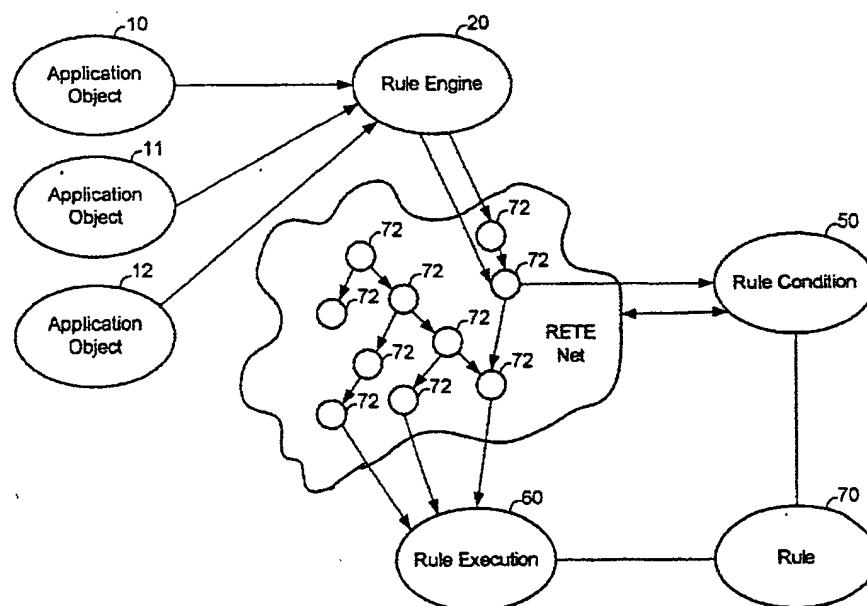


(57) Abstract: A Java rule engine framework is provided that permits a rule engine (20) to be called based upon events captured from objects (10, 11, 12) without requiring object programmers to explicitly insert hooks for calling the rule engine (20) within the objects (10, 11, 12). A business-to-business electronic marketplace is also provide with the Java rule engine framework for acting upon events occurring within objects (10, 11, 12). The framework includes a standard Java debugging interface adapted to accept events and a rule engine (20) to act upon such events.

# JAVA RULE ENGINE FRAMEWORK

## FIELD OF THE INVENTION

This invention relates to a Java rule engine framework.

## BACKGROUND

Rule engines are used to affect objects. In order to permit the application of rules to events occurring within objects, a rule engine must be capable of capturing events from the objects and programming against them. In an overall architecture of a rule engine that hosts an application, application objects call the rule engine. In order to function properly, the rule engine imposes certain requirements upon application objects. For example, rule engine must be able to receive the events from application objects that are to be used with a rule. Events that may be desirable to receive include field/property modification events, method calling events, object creation events and object deletion events. In order for a programmer to be able to code rules and conditions that can be used against application objects, the exact types of objects being used must be known to the programmer as well as details on the properties that are being used in the rules and conditions. The programmer must employ syntax that is able to express a wide variety of logistics while programming the rules and conditions.

Programmers of objects explicitly program hooks into the objects that reach out of the object and engage the rule engine when a specific event occurs through either programming the objects to a predefined object model or to meet the API's requirements.

Programmers have had to be aware of what the rule engines are looking for, spend time coding in the appropriate hooks and adhere to the syntax supported by the object model or API. There exists a need for a framework that would permit the application of rules through a rule engine without the need for the explicit insertions of hooks within objects.

# SUMMARY OF THE INVENTION

In an aspect, the invention features a rule engine framework including a rule engine for applying a rule against an object upon an occurrence of an event within the object, a debugging interface for detecting events from the object and

5    reporting the events to the rule engine, and an event handler thread for obtaining the event through the debugging interface and providing the event to the rule engine.

One or more or the following may be included. The rule engine and the object may be programmed in Java. The rule engine may apply the rule through a generation of a RETE Net.

10    The rule engine may also include a logging thread for logging the event, and a database for storing the logged event. The rule engine may include a rule callback thread for invoking a method on the object to affect a change within the object based upon an application of the rule to the event.

In another aspect, the invention features a method of capturing an

15    event from an object and acting upon the event without requiring the explicit insertion of hooks into an object including capturing the event through a debugging interface, providing the event to a rule engine, and applying a rule to the event.

One or more of the following may be included. The rule engine and the objects may be programmed in Java. Applying may include creating a RETE net.

20    The method may include logging the event on a database and/or affecting the object based upon the application of the rule.

In another aspect, the invention features an electronic marketplace having a rule engine framework including an object related to electronic commerce, a rule engine for applying a rule against the object upon an occurrence of an event

25    within the object, a debugging interface for detecting events from the object and reporting the events to the rule engine, and an event handler thread for obtaining the event through the debugging interface and providing the event to the rule engine.

One or more of the following may be included. The rule engine and the object may be programmed in Java. The rule engine may apply the rule through

30    the creation of a RETE Net.

The marketplace may also include a logging thread for logging the event, and a database for storing the logged event. The marketplace may include a rule callback thread for invoking a method on the object to affect a change within the object based upon the application of the rule to the event.

Embodiments of the invention may have one or more of the following advantages.

A Java rule engine framework permits the capture of events from Java objects themselves and the recording and/or processing of such events independent of application programming or object model.

An electronic business-to-business marketplace is provided having a generic Java rule engine framework that permits the capture of events from Java objects themselves and the recording and/or processing of such events independent of application programming or object model.

A rule engine is permitted to be called upon the occurrence of an event within a Java object without requiring the explicit insertion of hooks into the object for invoking the rule.

The rule engine is permitted to be called upon the occurrence of an event within a Java object on an electronic business-to-business marketplace without requiring the explicit insertion of hooks into the object for invoking the rule.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of an architecture of a rule engine.

FIG. 2 is a block diagram of a Java rule engine framework.

FIG. 3 is a block diagram of an electronic marketplace having the Java rule engine framework of FIG. 2.

# DETAILED DESCRIPTION

Referring to FIG. 1, an overall architecture 5 of a rule engine 20 that hosts an application is shown. Application objects 10, 11 and 12 call rule engine 20 upon the occurrence of predefined events. Rule engine 20 evaluates rule conditions, such as rule condition 50, to optimize when the rule should even be considered for execution by building RETE NET 30 using rule options expressed by the user. Rule 70 is accessed, as is rule condition 50. The rule is then executed as shown in box 60.

In general, a RETE NET, such as RETE NET 30, is a data structure that is used in the implementation of production systems. Much of its efficiency results from taking advantage of temporal redundancy by storing partial matches. Because a database is expected to change only slowly, this allows rapid matching as new data arrives. The RETE NET can be considered as a discrimination net augmented with memory at the individual nodes. Working memory elements that are propagated through the net are formed into lists of tokens corresponding to partial matches against a rule's left-hand side. A token of length N represents the working memory elements corresponding to the first N positive condition elements in the rule's right-hand side; all variable bindings within a token are guaranteed to be consistent. These tokens are stored in memory nodes 72 within the network. The RETE NET has two types of beta nodes that require that state be preserved. Simplifying somewhat, AND nodes concatenate tokens together and ensure consistent bindings of variables. NOT nodes serve as gates and pass tokens only if they have no consistent variable binding with any working memory element matching a negated condition element. Each beta node has two inputs, one representing an incoming working memory element and one representing an incoming token consisting of a list of previously matched elements. Because each element arriving at one input of the node must be compared with each element arriving at the other input, all tokens passing through a beta node are stored in either a left-hand or right-hand memory.

In order to function properly, rule engine 20 imposes certain requirements upon application objects 10, 11 and 12. For example, rule engine 20 must be able to receive the events from application objects 10, 11 and 12 that are to be used with rule 70. Events that may be desirable to receive include field/property modification events, method calling events, object creation events and object deletion

events.

In order for a programmer to be able to code rules and conditions, such as rule 70 and rule condition 50, which can be used in conjunction with application objects 10, 11 and 12, the exact types of objects being used must be known to a

5      programmer as well as details on the properties that are being used in the rules and conditions. The programmer employs syntax that is able to express a wide variety of logistics while programming the rules and conditions.

Programmers have methods with which to accomplish this. A first method is to have the rule engine define an Application Program Interface (API) for

10     the events that it requires. Under this scenario, the application objects would fire such events explicitly when they occur. An API, sometimes referred to as an application programming interface, is the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.

15     A second method is to have the rule engine define an object model that internally fires events automatically. Programmers of such objects would then have to ensure that each application object would adhere to the object model.

In either of these methods, programmers of objects have to explicitly include program hooks into the objects that would reach out of the object and engage

20     the rule engine when a specific event occurred through either programming the objects to a predefined object model or to meet the API's requirements. Thus, programmers have to be aware of what the rule engines are looking for, spend time coding in the appropriate hooks and adhere to the syntax supported by the object model or API. If an unforeseen rule is developed for use, the programming of the

25     objects has to be revisited in order to update the existing hooks or add new hooks to account for the new rule. This causes delays in deployment of needed services due to more time-intensive programming and the possibility of needed reprogramming.

The number of Java-based applications has grown considerably along with the growth of the Internet. In certain systems, events occurring within Java

30     objects may cause a desire to take some action. One such system is a business-to-business electronic marketplace. For instance, in an electronic marketplace a shopping

basket may be a Java object. When a method is executed that adds an item to that shopping basket, it may be desirable to cause some action to occur through the use of a rule engine, such as offering additional items to the shopper for possible addition to the shopping basket that relate to the newly added item.

5          Ideally, events could be captured from the Java objects themselves and then be processed independent of any application programming or any object model. As is described below, this can be accomplished through the use of debugging interfaces.

JVMDI (Java Virtual Machine Debugger Interface) is a Sun
10    Microsystems specification for debugging. The JVMDI prescribes that any JVM (Java Virtual Machine) that can be debugged adhere to a JPDA (Java Platform Debugger Architecture) specification.

A VM (Virtual Machine) supporting the JDPA architecture implements a JVMDI interface that defines the services that the JVM provides. It is possible to
15    hook onto these interfaces and get events from the VM. This is done in the native language using a JNI (Java Native Interface). The JNI is a Java programming interface, or API, that allows developers to access the languages of a host system and determine the way Java integrates with native code. VM is a term used by Sun Microsystems, developers of the Java programming language and runtime
20    environment, to describe software that acts as an interface between complier Java binary code and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a Java virtual machine has been provided for a platform, any Java program (which, after compilation, is called bytecode) can run on that platform. Java was designed to allow application programs to be built that could
25    be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. Java's virtual machine makes this possible. The Java virtual machine specification defines an abstract rather than a real "machine" (or processor) and specifies an instruction set, a set of registers, a stack, a "garbage heap," and a method area. The real implementation of this abstract or logically
30    defined processor can be in other code that is recognized by the real processor or be built into the microprocessor.

The output of "compiling" a Java source program (a set of Java

language statements) is called bytecode. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real microprocessor instruction) or the bytecode can be compiled further for the real microprocessor using what is called a just-in-time compiler.

5        The back-end for the JDWP (Java Debug Wire Protocol) implements this interface and the JDWP uses a transport mechanism, either Shared Memory or Sockets, to transport these events to another JVM that is debugging the current JVM via the JDI (Java Debug Interface) specification.

         The JDI defines numerous event requests. Some of those event requests support events that the rule engine requires. These events include, for
10      example, the following: 1) Com. sun. jdi.event.ModificationWatchpointEvent, which defines an event that the JVM generates when a particular field of an object is modified; 2) Com. sun. jdi. event. BreakpointEvent, which defines a breakpoint in the JVM execution (for use in a framework 100, it can define when a particular method or all methods have been entered or exited); Com. sun. jdi. event. ClassPrepareEvent,
15      which occurs when the class of the object is first being loaded; Com. sun. jdi. event. VMDeathEvent, which is fired when the Target JVM dies; Com. sun. jdi. event. VMDisconnectEvent, which is fired when the target JVM gets disconnected; and Com. sun. jdi. event. VMStartEvent, which is fired when the target JVM starts. Other
20      events that the JVM fires may be useful as well.

         Referring to FIG. 2, a Java rule engine framework 100 is shown. Hosted/Target JVM 160, the Java object from which events are to be captured, is connected to JVMDI 155. When an event occurs within Hosted/Target JVM 160, the event is fired to JVMDI 155. For instance, an item added to a shopping basket object
25      may cause such an event.

         JVMDI 155 then provides the event to JDWP 150. To tie into JVMDI 155, JDWP 150 is implemented in the native language.

         JDWP 150 conveys the event to JDI (Java Debug Interface) 145. JDWP 150 utilizes a transport mechanism, such as shared memory or sockets, to
30      convey the event. JDI 145 provides the event to the event handler thread 120, completing the event capture. If the event is desired to be logged as a business event,

logging thread 130 is also called. Logging thread 130 then records the event in business event capture database 140.

From the event handler thread 120, the rule engine 110 is called in main thread 125 and event definition database 135 is accessed as needed. Java
5   provides a mechanism for dynamic class loading, so the rules can be loaded when needed.

A rule written in Java may need to adhere to an interface that the rule engine 110 recognizes, but the internal code would be Java and the rule developer would write in Java. Rules are executed based on their activation by user options
10  and when events occur on objects in an application. Conditions of the rules may need to be split into individual conditions, and their contexts bound individually to build the NET.

Upon the application of the rule as discussed above, if a change is affected in the hosted/target JVM 160, rule callback thread 115 invokes the method to
15  cause the change in hosted/target JVM 160 through an API 165 defined through the JDI interface 145.

Referring to FIG. 3, an electronic marketplace 100 using the Java rule engine framework 100 of FIG. 2 is shown. Electronic marketplace members 205, 210 and 215 and outside member 235 are connected to a network 200, such as the Internet.
20  The network 200 also is linked to services 225, 230. In embodiments, multiple networks are used. Hosted/target JVM Object 160 within electronic marketplace 100 may be an object used in effectuating electronic commerce.

Other embodiments are within the scope of the following claims.

What is claimed is:

25

1.      A rule engine framework comprising:

a rule engine for applying a rule against an object upon an occurrence of an event within the object;

a debugging interface for detecting events from the object and reporting the events to the rule engine; and

an event handler thread for obtaining the event through the debugging interface and providing the event to the rule engine.

2.      The rule engine framework of claim 1 in which the rule engine and the object are programmed in Java.

3.      The rule engine framework of claim 1 in which the rule engine applies the rule through a generation of a RETE Net.

4.      The rule engine framework of claim 1 further comprising:

a logging thread for logging the event; and

a database for storing the logged event.

5.      The rule engine framework of claim 1 further comprising a rule callback thread for invoking a method on the object to affect a change within the object based upon an application of the rule to the event.

6.      A method of capturing an event from an object and acting upon the event without requiring the explicit insertion of hooks into an object comprising:

capturing the event through a debugging interface;

providing the event to a rule engine; and

applying a rule to the event.

7.      The method of claim 6 in which the rule engine and the objects are programmed in Java.

8.      The method of 6 in which applying comprises creating a RETE net.

9.      The method of claim 6 further comprising logging the event on a

9

database.

10.    A method of claim 6 further comprising affecting the object based upon the application of the rule.

11.    An electronic marketplace having a rule engine framework comprising:

an object related to electronic commerce;

a rule engine for applying a rule against the object upon an occurrence of an event within the object;

a debugging interface for detecting events from the object and reporting the events to the rule engine; and

an event handler thread for obtaining the event through the debugging interface and providing the event to the rule engine.

12.    The electronic marketplace of claim 11 in which the rule engine and the object are programmed in Java.

13.    The electronic marketplace of claim 11 in which the rule engine applies the rule through the creation of a RETE Net.

14.    The electronic marketplace of claim 11 further comprising:

a logging thread for logging the event; and

a database for storing the logged event.

15.    The electronic marketplace of claim 11 further comprising a rule callback thread for invoking a method on the object to affect a change within the object based upon the application of the rule to the event.
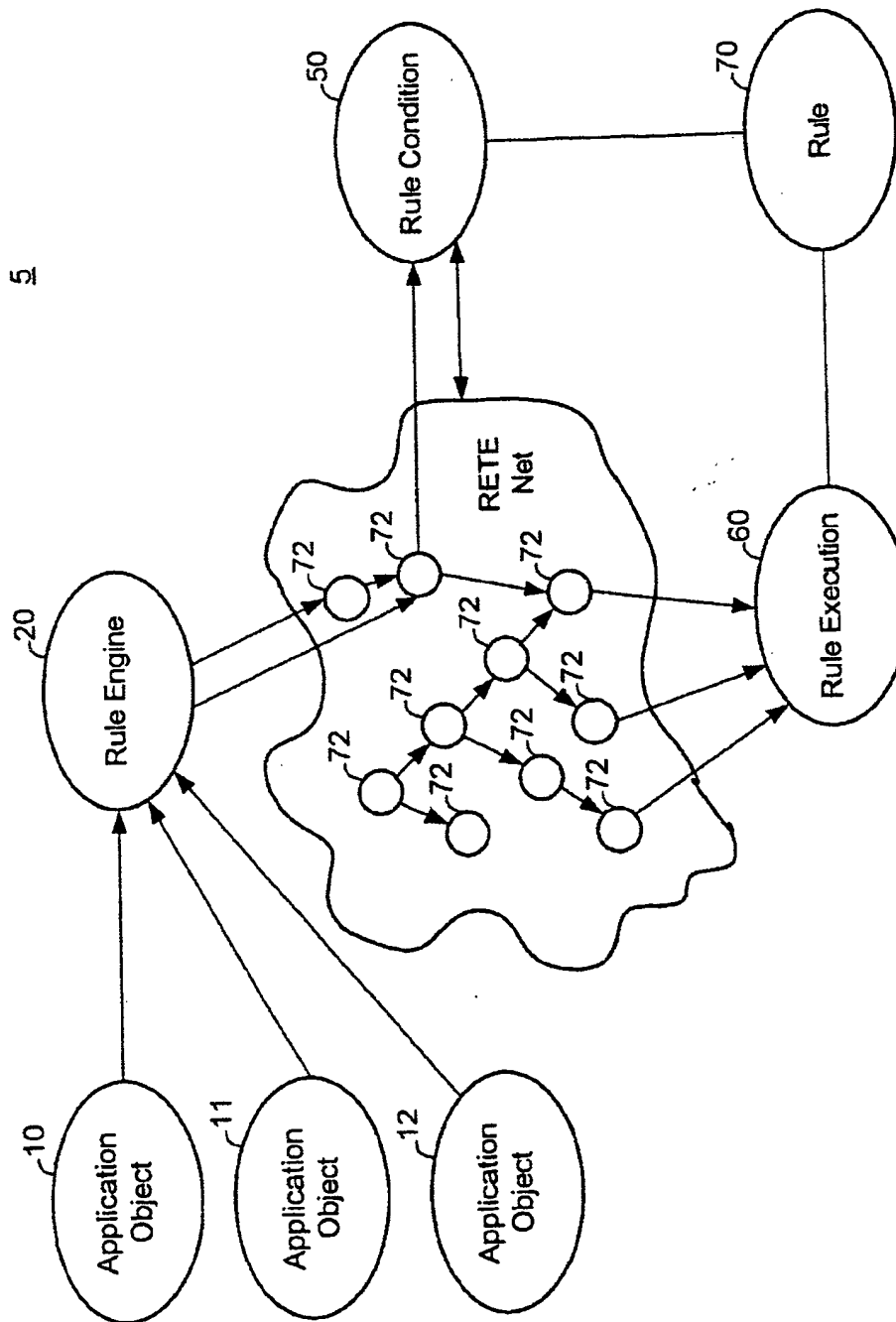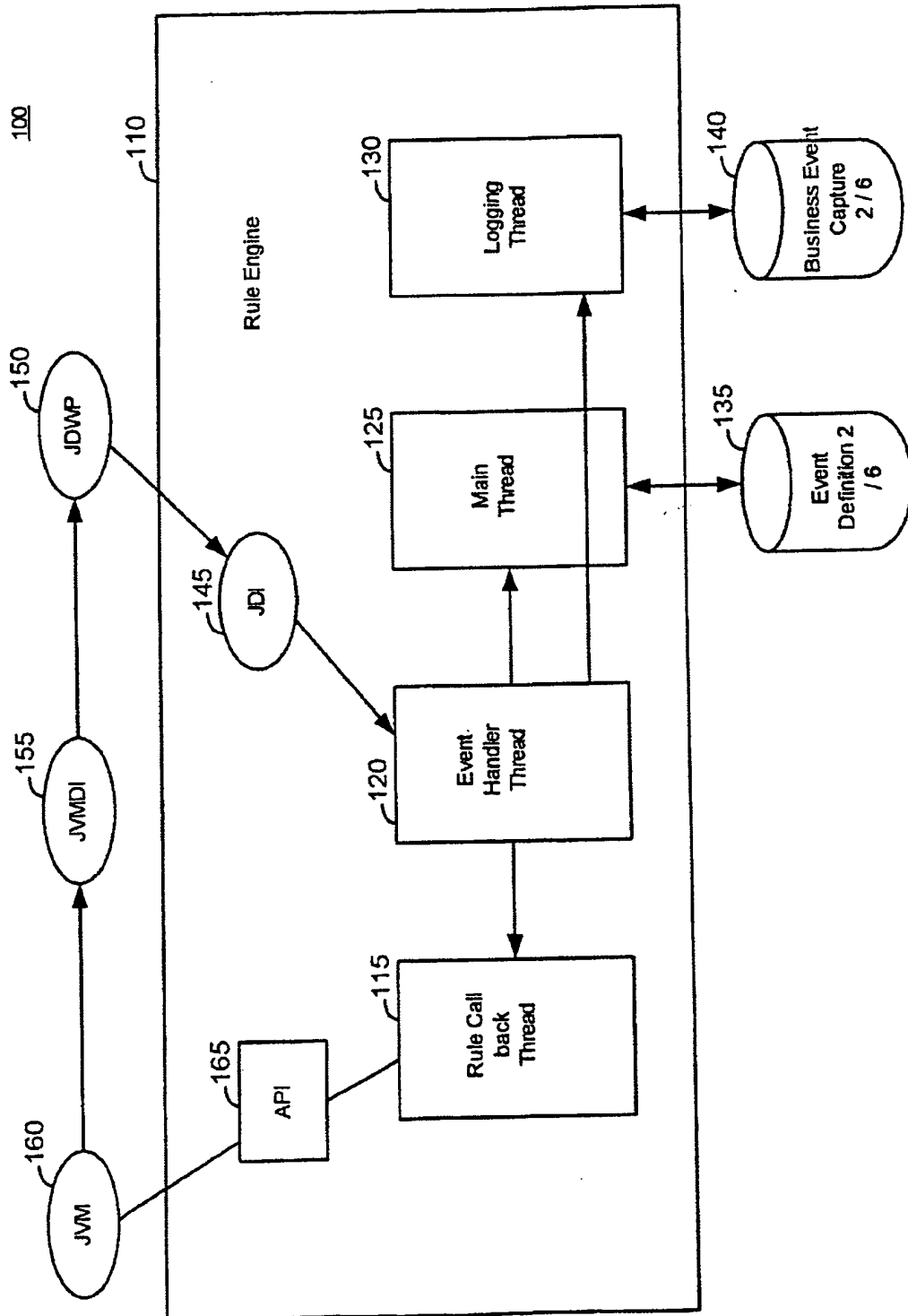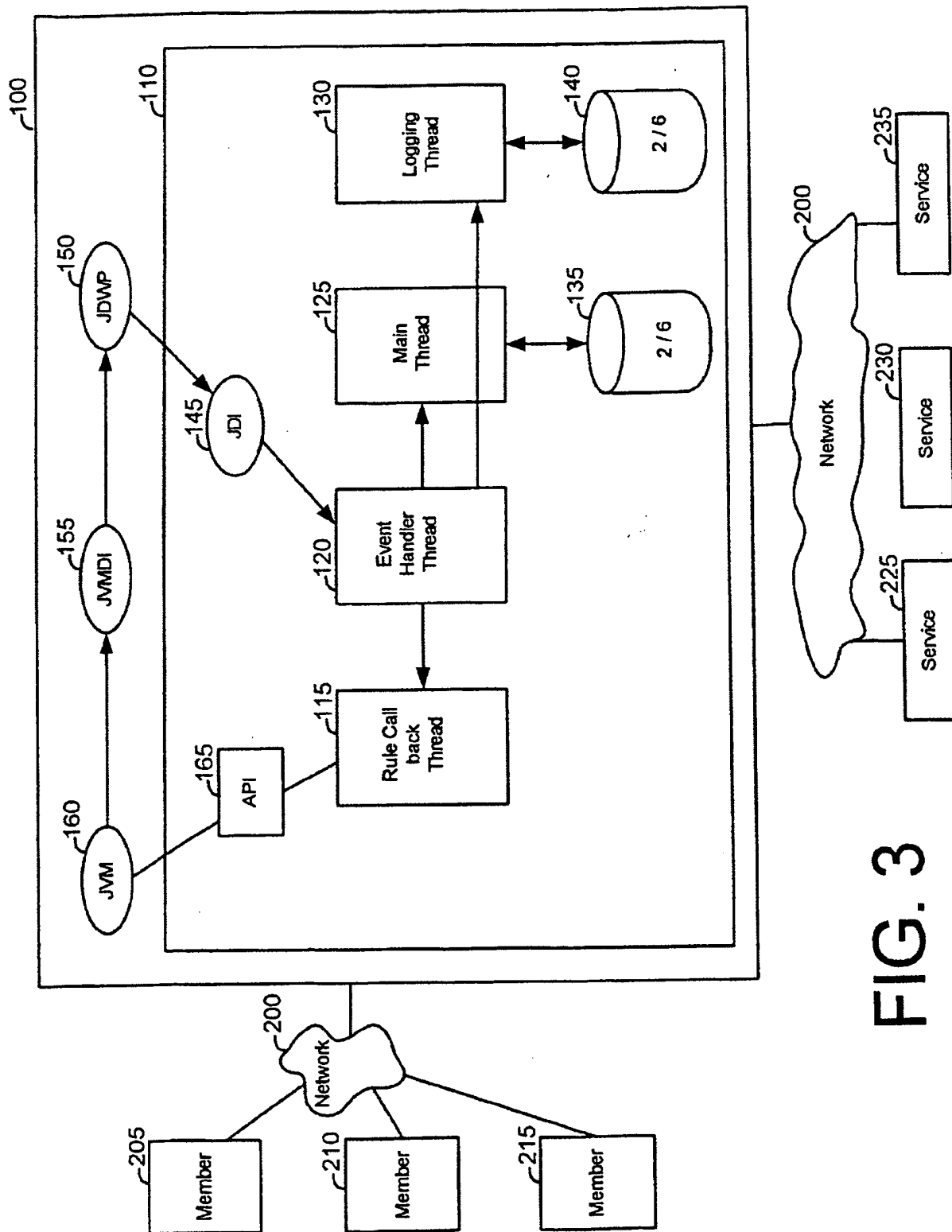
FIG. 1

FIG. 2

FIG. 3

# INTERNATIONAL SEARCH REPORT

International application No.

PCT/US02/19883

## A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 9/44

US CL : 717/124

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 717/124; 717/116,117,118; 714/2,25; 706/45,46,47,53; 709/217; 714/2,25

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST, ACM, STN, GOOGLE, IEEE

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| Y | US 6,018,732 A (BERTRAND et al) 25 JANUARY, 2000, Abstract, column 1 line 8-10, column 1 line 65-67, column 5 line 25-65, column 8 line 18-22, column 9 line 30-38, column 10 line 36-45, column 147 line 30-35 | 1-15 |
| Y | US 5,720,009 A (KIRK et al) 17 FEBRUARY, 1998, Abstract, column 1 line 10-15, column 2 line 15-30, column 3 line 19-24, column 4 line 53-55, column 11 line 12-14 | 1-15 |
| A | US 5,390,286 A (RAMAKRISHNA) 14 FEBRUARY, 1995, see entire document | 1-15 |
| A | US 5,179,633 A (BARABASH et al) 12 JANUARY, 1993, see entire document | 1-15 |

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 26 JULY 2002 | 20 AUG 2002 |

| Name and mailing address of the ISA/US<br>Commissioner of Patents and Trademarks<br>Box PCT<br>Washington, D.C. 20231<br>Facsimile No. (703) 305-3230 | Authorized officer<br><br>CHAMELI C. DAS<br><br>Telephone No. (703) 305-1339 |

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|-----------|----------------------------------------------------------------------------------|-----------------------|
| A | US 6,094,729 A (MANN) 25 JULY, 2000, see entire document | 1-15 |
| A | TITLE: A Tool for Internet-Oriented Knoledge Based Systems, author: Inder, ACM, March, 2000, see entire document | 1-15 |